D206 Performance Assessment

Author: Coots, Anthony
ID: 010958511
Date: 03/21/2024

Table of Contents:

Research Question

A: Question or Decision

"Of the factors provided, which factors predict readmission within a single month of discharge for the average patient?"

B: Required Variables

Variable Name	Data Type	Description	Example
CaseOrder	Identifier	Used for ordering the observations.	1
Customer_id	Identifier	Patient identifier value.	X110648
Interaction	Identifier	Hospital transaction identifier value.	e3b0a319-9e2e-4a23-8752-2fdc736c30f4
UID	Identifier	Additional unique identifier value. See Interaction.	8e866e2402ea8db6d0584209f714e
City	Qualitative/Categorical	City where patient holds residency.	Stoddard
State	Qualitative/Categorical	State where patient holds residency.	NH
County	Qualitative/Categorical	County where patient holds residency.	heshire
Zip	Qualitative/Categorical	Zipcode where patient holds residency.	68164
Lat	Quantitative/Numeric	Latitude coordinates of patient residency.	41.12489
Lng	Quantitative/Numeric	Longitude coordinates of patient residency.	-72.11417
Population	Quantitative/Numeric	General population within patient residency.	2951
Area	Qualitative/Categorical	General area of patient residency.	Rural
Timezone	Qualitative/Categorical	Time zone of patient	America/Los_Angeles

Variable Name	Data Type	Description	Example
		residency.	
Job	Qualitative/Categorical	Working job title of patient.	Police officer
Children	Quantitative/Numeric	The integer amount of children a patient has.	6
Age	Quantitative/Numeric	The integer age of a patient.	22
Education	Qualitative/Categorical	Level of education disclosed by patient.	Master's Degree
Employment	Qualitative/Categorical	Means of employment level of patient.	Full Time
Income	Quantitative/Numeric	Yearly income disclosed by patient.	62054.63
Marital	Qualitative/Categorical	Marital status disclosed by patient.	Married
Gender	Qualitative/Categorical	Gender indentification disclosed by patient.	Male
ReAdmis	Qualitative/Categorical	Whether or not the patient has been readmitted a month since initial admission.	No
VitD_levels	Quantitative/Numeric	Measure of patient's vitamin D level (ng/mL.)	17.80233
Doc_visits	Quantitative/Numeric	Measure of primary doctor visits in initial admit.	4
Full_meals_eaten	Quantitative/Numeric	Measure of full meals ate	2

Variable Name	Data Type	Description	Example
		while admitted.	
VitD_supp	Quantitative/Numeric	Measure of vitamin D supplements given to patient.	1
Soft_drink	Qualitative/Categorical	Whether or not patient regularly consumes soft drinks.	Yes
Initial_admin	Qualitative/Categorical	General reason for visit.	Elective Admission
HighBlood	Qualitative/Categorical	Patient has hypertension.	No
Stroke	Qualitative/Categorical	Patient has had a stroke.	No
Complication_risk	Qualitative/Categorical	Patient level of complication risk.	Medium
Overweight	Qualitative/Categorical	Patient is overweight.	1
Arthritis	Qualitative/Categorical	Patient has arthritis.	No
Diabetes	Qualitative/Categorical	Patient has diabetes.	1
Hyperlipidemia	Qualitative/Categorical	Patient has hyperlipidemia.	No
BackPain	Qualitative/Categorical	Patient has back pain.	Yes
Anxiety	Qualitative/Categorical	Patient has anxiety.	1
Allergic_rhinitis	Qualitative/Categorical	Patient has allergic rhinitis.	No
Reflux_esophagitis	Qualitative/Categorical	Patient has reflux esophagitis.	No
Asthma	Qualitative/Categorical	Patient has asthma.	Yes
Services	Qualitative/Categorical	Type of work patient has	CT Scan

Variable Name	Data Type	Description	Example
		had done while admitted.	
Initial_days	Quantitative/Numeric	The numeric number of days of admission.	9.05821
TotalCharge	Quantitative/Numeric	Total daily charge to patient not including additional charges.	3191.048774
Additional_charges	Quantitative/Numeric	Patient charges for specialized treatments.	16815.5136
Item1	Qualitative/Categorical	Survey result for speed of admission.	3
Item2	Qualitative/Categorical	Survey result for speed of treatment.	3
Item3	Qualitative/Categorical	Survey result for speed of visit.	2
Item4	Qualitative/Categorical	Survey result for reliability.	1
Item5	Qualitative/Categorical	Survey result for options available.	6
Item6	Qualitative/Categorical	Survey result for time of treatment.	3
Item7	Qualitative/Categorical	Survey result for staff friendliness.	3
Item8	Qualitative/Categorical	Survey result for active listening.	6

Data-Cleaning Plan

C1: Plan to Assess Quality of Data

The plan to assess the quality of the medical raw dataset includes the detection of duplicates, missing values, outliers, and the need for re-expression of categorical values. Duplicates are detected using the df.duplicated().value_counts() function in a dataframe 'df', which indicates the number of 'True' and 'False' values corresponding distinct and unique rows, where 'False' is not duplicates and vice versa. The df.isnull().sum() function is utilized to count null (NaN) values across variables, identifying columns that may require data imputation. The missingno.matrix(df) function from the missingno library, will be employed to display NaN and non-NaN values within the dataframe, providing a visualization of data completeness. Outliers are identified through the sns.boxplot() function from seaborn, which generates boxplots. This method is chosen for its effectiveness in highlighting outliers. The re-expression of categorical values, especially for columns with responses like 'Yes' or 'No', is addressed by identifying relevant columns with df.columns[df.isin(['Yes', 'No']).any()].tolist(), preparing the dataset for further analysis by ensuring categorical data is represented accurately, typically as ones or zeros. Additionally, data frame functions, such as astype will be used to re-express certain variables when appropriate post one-hot and ordinal encoding when necessary.

C2: Justification of Approach

Finding duplicates is important for ensuring accuracy, reducing bias, and maintaining consistency within this dataset. Duplicates can affect data by exaggerating certain conditions or demographic values. This directly impacts the analysis, in a bias manner. For instance, if the dataset counts a duplicate record of a diabetic individual, the ratio of diabetic to non-diabetic individuals could appear misleadingly even. The method outlined in Section C1 addresses this issue by checking for duplicate rows. If duplicates were to exist, the output would display of the function call would state 'True' alongside the count of such records. However, the analysis confirms that all records are unique, as indicated by 'False' for each of the ten thousand observations, verifying the absence of duplicates. Ultimately, this verification step is essential for ensuring the dataset's integrity.

Detecting missing values is an important data cleaning step. Removing observations due to the absence of values in certain variables while is straightforward, this action risks discarding valuable information in other variables. The df.isnull().sum().sort_values(ascending = False) function is used for identify missing values involves producing the name and count of each column, detailing the count of missing entries within each column in the data set. Additionally, 'missingno' matrix is used for visualization, offering a picture of data completeness for each column in the data set. These methods not only highlight areas of concern but also guides the decision-making process on how best to handle the missing data. By imputation, the route to address the removal and reassignment of outliers is mapped.

Among various visualization techniques like histograms and z-scores, boxplots stand out when it comes to identifying outliers in variables. Utilizing Seaborn, the popular Python library for visualization, these boxplots come straightforward and efficient. To detect outliers, the process involves a simple function call for each variable. This approach not only simplifies the detection of outliers but also provides a clear visual of the data's distribution and the values that could either be true data points for representation or potential data entry errors and etcetera, facilitating a better analysis.

The function df.columns[df.isin(['Yes', 'No']).any()].tolist() will pinpoint which data frame columns contain binary categorical values expressed as 'Yes' or 'No', similarly 1 or 0. Consistency in data representation impacts data analysis specifically the data cleaning phase. The columns of categorical values can require conversion to a numerical format (e.g., 'Yes' to '1' and 'No' to '0') to better assist the cleaning phase, which typically handle numerical input for outlier detection, etc. This function operates by examining each element within the data frame for a column and value match with 'Yes' or 'No', and then applies .any() to each column to check for at least one occurrence of these values. These columns are then put into a list using .tolist(). By doing this, the columns needing re-expression are now all containing binary categorical values, which ultimately assists the other parts of this phase, such as detecting and handling outliers. On top of binary categorical values, columns that represent numeric values will be appropriately address such that, income has no more than two places right of the decimal, or that columns like Children are a whole number. Additionally, multiple columns that are either nominal or ordinal will be made their own boolean variables, via one-hot or ordinal encoding.

C3: Justification of Tools

In this project, the language of choice, ‘Python’ is selected for library of (programming) libraries, significantly assisting in the data cleaning phase. Libraries such as matplotlib and seaborn for visualizations, enabling the creation of both simple and intermediate graphs. Pandas, a great tool for data manipulation, was used for cleaning to visualization with the data frame. Numpy was used for its’ advanced mathematical functions for dimensional arrays, while the os library streamlined accessing the current working directory. The statistics library provided basic statistical analyses, and the warnings library was employed to maintain a clear output by suppressing messages that are not necessarily essential for this project. Collectively, these libraries showcase Python's effective data cleaning capability.

C4: Provide the Code

Install missingno if not installed.

In [103…
```
pip install missingno
```

```
Requirement already satisfied: missingno in c:\users\antho\anaconda3\lib\site-pac
kages (0.5.2)
Requirement already satisfied: numpy in c:\users\antho\anaconda3\lib\site-package
s (from missingno) (1.26.4)
Requirement already satisfied: matplotlib in c:\users\antho\anaconda3\lib\site-pa
ckages (from missingno) (3.8.0)
Requirement already satisfied: scipy in c:\users\antho\anaconda3\lib\site-package
s (from missingno) (1.11.4)
Requirement already satisfied: seaborn in c:\users\antho\anaconda3\lib\site-packa
ges (from missingno) (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\antho\anaconda3\lib\s
ite-packages (from matplotlib->missingno) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\antho\anaconda3\lib\site-
packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\antho\anaconda3\lib
\site-packages (from matplotlib->missingno) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\antho\anaconda3\lib
\site-packages (from matplotlib->missingno) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\antho\anaconda3\lib\si
te-packages (from matplotlib->missingno) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\antho\anaconda3\lib\site
-packages (from matplotlib->missingno) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\antho\anaconda3\lib\s
ite-packages (from matplotlib->missingno) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\antho\anaconda3\l
ib\site-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: pandas>=0.25 in c:\users\antho\anaconda3\lib\site-
packages (from seaborn->missingno) (2.1.4)
Requirement already satisfied: pytz>=2020.1 in c:\users\antho\anaconda3\lib\site-
packages (from pandas>=0.25->seaborn->missingno) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\antho\anaconda3\lib\sit
e-packages (from pandas>=0.25->seaborn->missingno) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\antho\anaconda3\lib\site-pack
ages (from python-dateutil>=2.7->matplotlib->missingno) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [104…

```python
import importlib.util
import matplotlib.patches as pat
import matplotlib.pyplot as plt
import missingno as msno
import numpy as np
import os
import pandas as pd
import seaborn as sns
from scipy import stats
from sklearn.decomposition import PCA
import statistics
import warnings

# What is my current working directory?
print(os.getcwd())

# Read csv into data frame.
df = pd.read_csv('medical_raw_data.csv')
```

```
C:\Users\antho\OneDrive\Desktop\School\WGU\Data Analytics, M.S\D206\e9d8sm5uf8df7
5k650df
```

Data frame 'df'

In [105…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 53 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          10000 non-null  int64  
 1   CaseOrder           10000 non-null  int64  
 2   Customer_id         10000 non-null  object 
 3   Interaction         10000 non-null  object 
 4   UID                 10000 non-null  object 
 5   City                10000 non-null  object 
 6   State               10000 non-null  object 
 7   County              10000 non-null  object 
 8   Zip                 10000 non-null  int64  
 9   Lat                 10000 non-null  float64
 10  Lng                 10000 non-null  float64
 11  Population          10000 non-null  int64  
 12  Area                10000 non-null  object 
 13  Timezone            10000 non-null  object 
 14  Job                 10000 non-null  object 
 15  Children            7412 non-null   float64
 16  Age                 7586 non-null   float64
 17  Education           10000 non-null  object 
 18  Employment          10000 non-null  object 
 19  Income              7536 non-null   float64
 20  Marital             10000 non-null  object 
 21  Gender              10000 non-null  object 
 22  ReAdmis             10000 non-null  object 
 23  VitD_levels         10000 non-null  float64
 24  Doc_visits          10000 non-null  int64  
 25  Full_meals_eaten    10000 non-null  int64  
 26  VitD_supp           10000 non-null  int64  
 27  Soft_drink          7533 non-null   object 
 28  Initial_admin       10000 non-null  object 
 29  HighBlood           10000 non-null  object 
 30  Stroke              10000 non-null  object 
 31  Complication_risk   10000 non-null  object 
 32  Overweight          9018 non-null   float64
 33  Arthritis           10000 non-null  object 
 34  Diabetes            10000 non-null  object 
 35  Hyperlipidemia      10000 non-null  object 
 36  BackPain            10000 non-null  object 
 37  Anxiety             9016 non-null   float64
 38  Allergic_rhinitis   10000 non-null  object 
 39  Reflux_esophagitis  10000 non-null  object 
 40  Asthma              10000 non-null  object 
 41  Services            10000 non-null  object 
 42  Initial_days        8944 non-null   float64
 43  TotalCharge         10000 non-null  float64
 44  Additional_charges  10000 non-null  float64
 45  Item1               10000 non-null  int64  
 46  Item2               10000 non-null  int64  
 47  Item3               10000 non-null  int64  
 48  Item4               10000 non-null  int64  
 49  Item5               10000 non-null  int64  
 50  Item6               10000 non-null  int64  
```

```
 51  Item7               10000 non-null  int64
 52  Item8               10000 non-null  int64
dtypes: float64(11), int64(15), object(27)
memory usage: 4.0+ MB
```

Duplicates

In [106…

```
# Print count of duplicated observations.
print(df.duplicated().value_counts())
```

```
False    10000
Name: count, dtype: int64
```

In [107…

```
for col in df:
    if df.columns.tolist().index(col) > 0:
        print(df[col].duplicated().value_counts())
```

```
CaseOrder
False    10000
Name: count, dtype: int64
Customer_id
False    10000
Name: count, dtype: int64
Interaction
False    10000
Name: count, dtype: int64
UID
False    10000
Name: count, dtype: int64
City
False    6072
True     3928
Name: count, dtype: int64
State
True     9948
False      52
Name: count, dtype: int64
County
True     8393
False    1607
Name: count, dtype: int64
Zip
False    8612
True     1388
Name: count, dtype: int64
Lat
False    8588
True     1412
Name: count, dtype: int64
Lng
False    8601
True     1399
Name: count, dtype: int64
Population
False    5951
True     4049
Name: count, dtype: int64
Area
True     9997
False       3
Name: count, dtype: int64
Timezone
True     9974
False      26
Name: count, dtype: int64
Job
True     9361
False     639
Name: count, dtype: int64
Children
True     9988
False      12
Name: count, dtype: int64
```

```
Age
True     9927
False      73
Name: count, dtype: int64
Education
True     9988
False      12
Name: count, dtype: int64
Employment
True     9995
False       5
Name: count, dtype: int64
Income
False    7532
True     2468
Name: count, dtype: int64
Marital
True     9995
False       5
Name: count, dtype: int64
Gender
True     9997
False       3
Name: count, dtype: int64
ReAdmis
True     9998
False       2
Name: count, dtype: int64
VitD_levels
False    10000
Name: count, dtype: int64
Doc_visits
True     9991
False       9
Name: count, dtype: int64
Full_meals_eaten
True     9992
False       8
Name: count, dtype: int64
VitD_supp
True     9994
False       6
Name: count, dtype: int64
Soft_drink
True     9997
False       3
Name: count, dtype: int64
Initial_admin
True     9997
False       3
Name: count, dtype: int64
HighBlood
True     9998
False       2
Name: count, dtype: int64
Stroke
```

```
True     9998
False       2
Name: count, dtype: int64
Complication_risk
True     9997
False       3
Name: count, dtype: int64
Overweight
True     9997
False       3
Name: count, dtype: int64
Arthritis
True     9998
False       2
Name: count, dtype: int64
Diabetes
True     9998
False       2
Name: count, dtype: int64
Hyperlipidemia
True     9998
False       2
Name: count, dtype: int64
BackPain
True     9998
False       2
Name: count, dtype: int64
Anxiety
True     9997
False       3
Name: count, dtype: int64
Allergic_rhinitis
True     9998
False       2
Name: count, dtype: int64
Reflux_esophagitis
True     9998
False       2
Name: count, dtype: int64
Asthma
True     9998
False       2
Name: count, dtype: int64
Services
True     9996
False       4
Name: count, dtype: int64
Initial_days
False    8945
True     1055
Name: count, dtype: int64
TotalCharge
False    10000
Name: count, dtype: int64
Additional_charges
False    8888
```

```
True      1112
Name: count, dtype: int64
Item1
True      9992
False        8
Name: count, dtype: int64
Item2
True      9993
False        7
Name: count, dtype: int64
Item3
True      9992
False        8
Name: count, dtype: int64
Item4
True      9993
False        7
Name: count, dtype: int64
Item5
True      9993
False        7
Name: count, dtype: int64
Item6
True      9993
False        7
Name: count, dtype: int64
Item7
True      9993
False        7
Name: count, dtype: int64
Item8
True      9993
False        7
Name: count, dtype: int64
```

Missing Values

In [108…

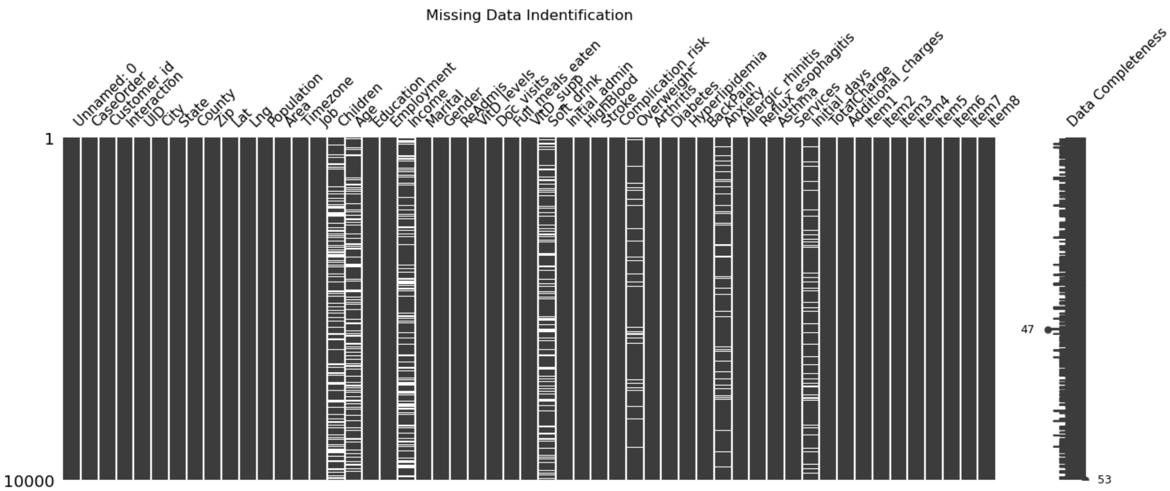
```
# Detect null values in dataset.
df.isnull().sum().sort_values(ascending = False)
```

Out[108…

```
Children              2588
Soft_drink            2467
Income                2464
Age                   2414
Initial_days          1056
Anxiety                984
Overweight             982
Stroke                   0
Complication_risk        0
Arthritis                0
Diabetes                 0
Hyperlipidemia           0
BackPain                 0
Allergic_rhinitis        0
Unnamed: 0               0
HighBlood                0
Asthma                   0
Services                 0
TotalCharge              0
Additional_charges       0
Item1                    0
Item2                    0
Item3                    0
Item4                    0
Item5                    0
Item6                    0
Item7                    0
Reflux_esophagitis       0
VitD_supp                0
Initial_admin            0
CaseOrder                0
Customer_id              0
Interaction              0
UID                      0
City                     0
State                    0
County                   0
Zip                      0
Lat                      0
Lng                      0
Population               0
Area                     0
Timezone                 0
Job                      0
Education                0
Employment               0
Marital                  0
Gender                   0
ReAdmis                  0
VitD_levels              0
Doc_visits               0
Full_meals_eaten         0
Item8                    0
dtype: int64
```

In [109…

```
# Detect missing values in dataset.
msno.matrix(df, figsize = (15, 5), fontsize = 11, labels = True)

plt.title('Missing Data Indentification')
plt.show()
```



In [110…

```
print('Outliers:\n')

# Find columns that have outliers (data points greater than or less than their re
for col in df:
   if ((df[col].dtypes == 'float64' or df[col].dtypes == 'int64')) and df.column
        Q1 = df[col].describe()['25%']
        Q3 = df[col].describe()['75%']
        IQR = Q3 - Q1
        whiskerL = (Q1 - (1.5 * IQR))
        whiskerR = (Q3 + (1.5 * IQR))
        whiskerL = (df[df[col] >= whiskerL][col].min())
        whiskerR = (df[df[col] <= whiskerR][col].max())
        if any(df[col] > whiskerR) or any(df[col] < whiskerL):
            out_count = ((df[col] > whiskerR).sum() + (df[col] < whiskerL).sum()
            status = 'True'
        else:
            status = 'False'
        if status == 'True':
            print(str(col) + ':' + ((25 - len(str(col)) - len(str(status))) * '
        elif status == 'False':
            print(str(col) + ':' + ((34 - len(str(col)) - len(str(status))) * '
   elif (df[col].dtypes != 'float64' or df[col].dtypes != 'int64') and df.column
        print(str(col) + ':' + ((43 - len(str(col))) * ' ') + 'Not Applicable.')
```

```
Outliers:

CaseOrder:                                   Not Applicable.
Customer_id:                                 Not Applicable.
Interaction:                                 Not Applicable.
UID:                                         Not Applicable.
City:                                        Not Applicable.
State:                                       Not Applicable.
County:                                      Not Applicable.
Zip:                                         Not Applicable.
Lat:                                         Not Applicable.
Lng:                                         Not Applicable.
Population:                 True, number of outliers: 855.
Area:                                        Not Applicable.
Timezone:                                    Not Applicable.
Job:                                         Not Applicable.
Children:                   True, number of outliers: 303.
Age:                         False, number of outliers: 0.
Education:                                   Not Applicable.
Employment:                                  Not Applicable.
Income:                     True, number of outliers: 252.
Marital:                                     Not Applicable.
Gender:                                      Not Applicable.
ReAdmis:                                     Not Applicable.
VitD_levels:                True, number of outliers: 534.
Doc_visits:                  False, number of outliers: 0.
Full_meals_eaten:             True, number of outliers: 8.
VitD_supp:                   True, number of outliers: 70.
Soft_drink:                                  Not Applicable.
Initial_admin:                               Not Applicable.
HighBlood:                                   Not Applicable.
Stroke:                                      Not Applicable.
Complication_risk:                           Not Applicable.
Overweight:                  False, number of outliers: 0.
Arthritis:                                   Not Applicable.
Diabetes:                                    Not Applicable.
Hyperlipidemia:                              Not Applicable.
BackPain:                                    Not Applicable.
Anxiety:                     False, number of outliers: 0.
Allergic_rhinitis:                           Not Applicable.
Reflux_esophagitis:                          Not Applicable.
Asthma:                                      Not Applicable.
Services:                                    Not Applicable.
Initial_days:                False, number of outliers: 0.
TotalCharge:                True, number of outliers: 466.
Additional_charges:         True, number of outliers: 424.
Item1:                      True, number of outliers: 449.
Item2:                      True, number of outliers: 429.
Item3:                      True, number of outliers: 443.
Item4:                      True, number of outliers: 450.
Item5:                      True, number of outliers: 443.
Item6:                      True, number of outliers: 443.
Item7:                      True, number of outliers: 438.
Item8:                      True, number of outliers: 442.
```

In [111…

```
# Detect needed re-expression of categorical variables.
print('Needs re-expression:\n')

# Logic for if column in data frame should be re-expressed.
for col in df:
    if df.columns.tolist().index(col) > 10:
        # Anxiety or Overweight contains 1 and 0 which can be recognized as boole
        if (col == 'Anxiety' or col == 'Overweight' or col == 'Timezone' or col =
            status = 'False'
        elif (df[col].dtypes == 'float64' or df[col].dtypes == 'int64'):
            status = 'False'
        elif (df[col].dtypes == 'object' and df[col].isin(['Yes', 'No']).any()) a
            status = 'True'
        else:
            status = 'True'
        print(str(col) + ':' + ((20 - len(str(col))) * ' ') + status)
```

```
Needs re-expression:

Population:          False
Area:                True
Timezone:            False
Job:                 False
Children:            False
Age:                 False
Education:           True
Employment:          True
Income:              False
Marital:             True
Gender:              True
ReAdmis:             True
VitD_levels:         False
Doc_visits:          False
Full_meals_eaten:    False
VitD_supp:           False
Soft_drink:          True
Initial_admin:       True
HighBlood:           True
Stroke:              True
Complication_risk:   True
Overweight:          False
Arthritis:           True
Diabetes:            True
Hyperlipidemia:      True
BackPain:            True
Anxiety:             False
Allergic_rhinitis:   True
Reflux_esophagitis:  True
Asthma:              True
Services:            True
Initial_days:        False
TotalCharge:         False
Additional_charges:  False
Item1:               False
Item2:               False
Item3:               False
Item4:               False
Item5:               False
Item6:               False
Item7:               False
Item8:               False
```

Data Cleaning

D1: Cleaning Findings

The following has been observerd during the assessment of the dataset which involved detecting duplicates, missing values, outliers and the need for re-expression of categorical variables:

- The column provided to the left of `CaseOrder`, `Unnamed: 0` is repetitive and thus will be removed before the production of the cleaned dataset.
- The are no duplicate observations considering all variables.
- Columns `CaseOrder`, `Customer_id`, `Interaction`, `UID` individually show no duplicates. There is no missing values for these variables and outliers or re-expression may not be appropriate with these data points as they do not have inherent order (nominal).

- ### Demographic Data:
  - Column `City` is allowed duplicates and has no missing values, additionally, the outliers and/or re-expression may be misleading in the scenario.
  - Column `State` is allowed duplicates and has no missing values, additionally like `City`, the outliers and/or re-expression may be misleading in the scenario.
  - Column `County` is allowed duplicates and has no missing values, like `City` and `State`, the outliers and/or re-expression may be misleading in the scenario.
  - Column `Zip` is allowed duplicates and has no missing values. The outliers and/or re-expression may be misleading in the scenario. NOTE: Nominal data point.
  - Column `Lat` is allowed duplicates and has no missing values. The outliers and/or re-expression may be misleading in the scenario.
  - Column `Lng` is allowed duplicates and has no missing values. The outliers and/or re-expression may be misleading in the scenario.
  - Column `Population` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.
  - Column `Area` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Timezone` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
    - **NOTE**: `Timezone` based on patient sign-up according to data dictionary, will not be queued to remove data.
  - Column `Age` is allowed duplicates and **is** missing values. Outliers have not been detected. The variable does not require re-expression as it is quantitative.

- **NOTE**: `Age` is also depicted as a float64 variable, this should be changed to an integer field.

- Column `Gender` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

  - **NOTE**: Only values such as `"Male"`, `"Female"` and `"Prefer not to answer"` exist yet the data dictionary states for `"Male"`, `"Female"` and `"Nonbinary."`

- Column `Education` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

- Column `Employment` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

- Column `Income` is allowed duplicates and **is** missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.

- Column `Marital` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

- Column `Job` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

- Column `Children` is allowed duplicates and **is** missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.

  - **NOTE**: `Children` is also depicted as a float64 variable, this should be changed to an integer field.

- **Hospitalization Data:**

  - Column `Services` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Initial_admin` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Initial_days` is allowed duplicates and **is** missing values. Outliers have not been detected. The variable does not require re-expression as it is quantitative.
  - Column `TotalCharge` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.

- Column `Additional_charges` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.

- **Medical Data:**
  - Column `ReAdmis` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `VitD_levels` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.
  - Column `Doc_visits` is allowed duplicates and has no missing values. Outliers have not been detected. The variable does not require re-expression as it is quantitative.
  - Column `Full_meals_eaten` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.
  - Column `VitD_supp` is allowed duplicates and has no missing values. Outliers have been detected and should be addressed. The variable does not require re-expression as it is quantitative.
  - Column `Soft_drink` is allowed duplicates and **is** missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `HighBlood` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Stroke` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Overweight` is allowed duplicates and **is** missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Arthritis` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Diabetes` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Hyperlipidemia` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `BackPain` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Anxiety` is allowed duplicates and **is** missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Allergic_rhinitis` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
  - Column `Reflux_esophagitis` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not

numeric.

- Column `Asthma` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.
- Column `Complication_risk` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric.

- **Survey Data:**
    - Column `Item1` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item2` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item3` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item4` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item5` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item6` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item7` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does not require re-expression as it is in ordinal format, the desirable format for mitigation code.
    - Column `Item8` is allowed duplicates and has no missing values. Outliers have not been detected as the variable is qualitative, not numeric. The variable does

not require re-expression as it is in ordinal format, the desirable format for mitigation code.

- **NOTE**: Naming conventions are not practical as they lack description and create poor documentation.

Variable Name	Duplicates?	Missing Values?	Outliers?	Needs Re-expression?
CaseOrder	0	0	N/A	N/A, Quantitative Identifier.
Customer_id	0	0	N/A	N/A, Quantitative Identifier.
Interaction	0	0	N/A	N/A, Qualitative Identifier.
UID	0	0	N/A	N/A, Qualitative Identifier.
City	3928	0	No, Categorical.	Nominal.***
State	9948	0	No, Categorical.	Nominal.***
County	8393	0	No, Categorical.	Nominal.***
Zip	1388	0	No, Categorical.	Nominal.***
Lat	1412	0	N/A	N, Quantitative.
Lng	1399	0	N/A	N, Quantitative.
Population	4049	0	855	N, Quantitative.
Area	9997	0	No, Categorical.	Nominal.***
Timezone	9974	0	No, Categorical.	Nominal.***
Job	9361	0	No, Categorical.	Nominal.***
Children	9988	2588	303	No, Quantitative.
Age	9927	2414	No	No, Quantitative.
Education	9988	0	No, Categorical.	Ordinal.^^^
Employment	9995	0	No, Categorical.	Nominal.***

Variable Name	Duplicates?	Missing Values?	Outliers?	Needs Re-expression?
Income	7532	2464	252	N, Quantitative.
Marital	9995	0	No, Categorical.	Nominal.***
Gender	9997	0	No, Categorical.	Nominal.***
ReAdmis	9998	0	No, Categorical.	Nominal.***
VitD_levels	0	0	534	No, Quantitative.
Doc_vists	9991	0	No, Categorical.	No, Quantitative.
Full_meals_eaten	9992	0	8	No, Quantitative.
VitD_supp	9994	0	70	N, Quantitative.
Soft_drink	9997	2467	No, Categorical.	Nominal.***
Initial_admin	9997	0	No, Categorical.	Nominal.***
HighBlood	9998	0	No, Categorical.	Nominal.***
Stroke	9998	0	No, Categorical.	Nominal.***
Complication_risk	9997	0	No, Categorical.	Ordinal.^^^
Overweight	9997	982	No, Categorical.	Nominal.***
Arthritis	9998	0	No, Categorical.	Nominal.***
Diabetes	9998	0	No, Categorical.	Nominal.***
Hyperlipidemia	9998	0	No, Categorical.	Nominal.***
BackPain	9998	0	No, Categorical.	Nominal.***
Anxiety	9997	984	No, Categorical.	Nominal.***
Allergic_rhinits	9998	0	No, Categorical.	Nominal.***
Reflux_esophagitis	9998	0	No,	Nominal.***

Variable Name	Duplicates?	Missing Values?	Outliers?	Needs Re-expression?
			Categorical.	
Asthma	9998	0	No, Categorical.	Nominal.***
Services	9996	0	No, Categorical.	Nominal.***
Initial_days	1055	1056	No	No, Quantitative.
TotalCharge	0	0	466	No, Quantitative.
Additional_charges	1112	0	424	No, Quantitative.
Item1	9992	0	No, Categorical.	Ordinal.^^^
Item2	9993	0	No, Categorical.	Ordinal.^^^
Item3	9992	0	No, Categorical.	Ordinal.^^^
Item4	9993	0	No, Categorical.	Ordinal.^^^
Item5	9993	0	No, Categorical.	Ordinal.^^^
Item6	9993	0	No, Categorical.	Ordinal.^^^
Item7	9993	0	No, Categorical.	Ordinal.^^^
Item8	9993	0	No, Categorical.	Ordinal.^^^

**** The re-expression of these variables may be unnecessary or far-fetched given the scenario. However, some variables could see a change in data type. (e.g., converting from an Object type to a Boolean type.)*

^^^ The re-expression of these variables will use Ordinal Encoding. Some fields such as Item1-Item8 are already expressed appropriately. Additionally, some variables could see a change in data type. (e.g., converting from an Object type to a Int type.)

D2: Justification of Mitigation Methods

The duplication mitigation process involves reiteratively scanning each column of the cleaned data frame using the function df_clean[col].duplicated().value_counts() within a loop to verify the amount of duplicates per variable. Although the findings should not be particularly remarkable to the diligent worker, it should be known that identification columns such as CaseOrder, Customer_id, Interaction or UID should have no duplicates whatsoever. Likewise, an evaluation of entire observations using df_clean.duplicated().value_counts() should confirm the absence of duplicates for total observations in the dataset/data frame.

The missingno library's matrix visualization, with the df.isnull().sum() method from pandas, works in the mitigation process in order to verify the final results. Additionally they assist with the presence of missing values within the data frame with the detection process. Referencing the techniques outlined in the WGU D206 course materials titled 'Detecting and Treating Missing Values', missing data is imputed using the data frame's .mean(), .median(), and .mode() functions, for the distribution characteristics of each variable. Specifically, .mean() is used for variables displaying a uniform distribution, .median() is applied to variables displaying skewness, and .mode() is chosen for distributions that were displaying bimodal picking one of the two hills to represent the missing values, as seen in boolean qualitative variables such as 'Soft_drink'.

Columns containing outliers are initially identified through Seaborn boxplot visualizations for verification. Subsequently, using DataFrame functions like .min() and .max() in parallel to Q1, Q3 and IQR calculations will find exactly what values are outliers as these are values out of range of the boxplot whiskers. Then after, the implementation will handle outliers by first replacing them with null values using NumPy's .where() function in order to imputate with the .median() function for fair input. Regardless, some outliers might persist to prevent introducing bias through excessive distortion of the dataset. The use of plots was crucial in confirming that the imputation of outliers was executed correctly. **NOTE**: "We do not check categorical variables for outliers...In fact, there is no concept of outliers in a categorical variable." An informative conversation I have had with Dr. Eric Straw.

Finally, variables are then examined to re-express nominal and ordinal characteristics among the variables. Notably, columns such as 'State', 'City', 'County', and 'Zip' will be grouped for simplification of location and subsequently one-hot encoded the 'State' column in the cleaned DataFrame. The 'Education' column will then be ordinal encoded to numerically represent educational levels, with the highest level indicating a Doctorate degree/Professional School degree. Additionally, a few categorical variables present a "Yes" and "No" response and will be converted to a boolean column, with 1s and 0s for clearer representation among all columns one-hot encoded or boolean. A re-expression also involves transforming age values from decimals to whole numbers to re-define the dataset’s wholeness.

D3: Summary of the Outcomes

The duplication process performed as expected in the justification, no duplicates were found in any of the identification variables, CaseOrder, Customer_id, Interaction or UID. Similarly no duplicates of observations were found. All together there was indication of duplicates among variables which should be appropriate such as Age, Children, etc. The missingno matrix in parallel to the df.isnull().sum() indicated 7 columns had missing values, that were then imputated and verified to not distort the data. The verification of missing data and the imputation process can be seen more descriptively in D4. The outliers were appropriately removed onceover and then displayed again in boxplots. Though outliers still exist, as mentioned in D2, they remain as part of the new boxplot range and will remain to not distort the data in a bias manner. Finally, columns were appropriately involved in data type conversions with the addition of 15 columns from one-hot encoding. Ordinal encoding was also performed to indicate level numerically upon the Complication_risk and Education columns.

D4: Mitigation Code

In [112…

```
# Create copy of data frame for cleaning.
df_clean = df.copy()
df_clean.drop(df_clean.columns[df_clean.columns.str.contains('Unnamed')], axis =
```

Mitigating Duplicates / Verification Duplicates

In [113…

```
# Print count of duplicated observations.
print(df_clean.duplicated().value_counts())
```

```
False    10000
Name: count, dtype: int64
```

In [114…

```
# Value counts of duplicates (are allowed for individual variables besides ident
for col in df_clean:
    print(df_clean[col].duplicated().value_counts())
```

```
CaseOrder
False    10000
Name: count, dtype: int64
Customer_id
False    10000
Name: count, dtype: int64
Interaction
False    10000
Name: count, dtype: int64
UID
False    10000
Name: count, dtype: int64
City
False    6072
True     3928
Name: count, dtype: int64
State
True     9948
False      52
Name: count, dtype: int64
County
True     8393
False    1607
Name: count, dtype: int64
Zip
False    8612
True     1388
Name: count, dtype: int64
Lat
False    8588
True     1412
Name: count, dtype: int64
Lng
False    8601
True     1399
Name: count, dtype: int64
Population
False    5951
True     4049
Name: count, dtype: int64
Area
True     9997
False       3
Name: count, dtype: int64
Timezone
True     9974
False      26
Name: count, dtype: int64
Job
True     9361
False     639
Name: count, dtype: int64
Children
True     9988
False      12
Name: count, dtype: int64
```

```
Age
True     9927
False      73
Name: count, dtype: int64
Education
True     9988
False      12
Name: count, dtype: int64
Employment
True     9995
False       5
Name: count, dtype: int64
Income
False    7532
True     2468
Name: count, dtype: int64
Marital
True     9995
False       5
Name: count, dtype: int64
Gender
True     9997
False       3
Name: count, dtype: int64
ReAdmis
True     9998
False       2
Name: count, dtype: int64
VitD_levels
False    10000
Name: count, dtype: int64
Doc_visits
True     9991
False       9
Name: count, dtype: int64
Full_meals_eaten
True     9992
False       8
Name: count, dtype: int64
VitD_supp
True     9994
False       6
Name: count, dtype: int64
Soft_drink
True     9997
False       3
Name: count, dtype: int64
Initial_admin
True     9997
False       3
Name: count, dtype: int64
HighBlood
True     9998
False       2
Name: count, dtype: int64
Stroke
```

```
True     9998
False       2
Name: count, dtype: int64
Complication_risk
True     9997
False       3
Name: count, dtype: int64
Overweight
True     9997
False       3
Name: count, dtype: int64
Arthritis
True     9998
False       2
Name: count, dtype: int64
Diabetes
True     9998
False       2
Name: count, dtype: int64
Hyperlipidemia
True     9998
False       2
Name: count, dtype: int64
BackPain
True     9998
False       2
Name: count, dtype: int64
Anxiety
True     9997
False       3
Name: count, dtype: int64
Allergic_rhinitis
True     9998
False       2
Name: count, dtype: int64
Reflux_esophagitis
True     9998
False       2
Name: count, dtype: int64
Asthma
True     9998
False       2
Name: count, dtype: int64
Services
True     9996
False       4
Name: count, dtype: int64
Initial_days
False    8945
True     1055
Name: count, dtype: int64
TotalCharge
False    10000
Name: count, dtype: int64
Additional_charges
False    8888
```

```
True     1112
Name: count, dtype: int64
Item1
True     9992
False       8
Name: count, dtype: int64
Item2
True     9993
False       7
Name: count, dtype: int64
Item3
True     9992
False       8
Name: count, dtype: int64
Item4
True     9993
False       7
Name: count, dtype: int64
Item5
True     9993
False       7
Name: count, dtype: int64
Item6
True     9993
False       7
Name: count, dtype: int64
Item7
True     9993
False       7
Name: count, dtype: int64
Item8
True     9993
False       7
Name: count, dtype: int64
```

Mitigating - Missing Values

In [115…

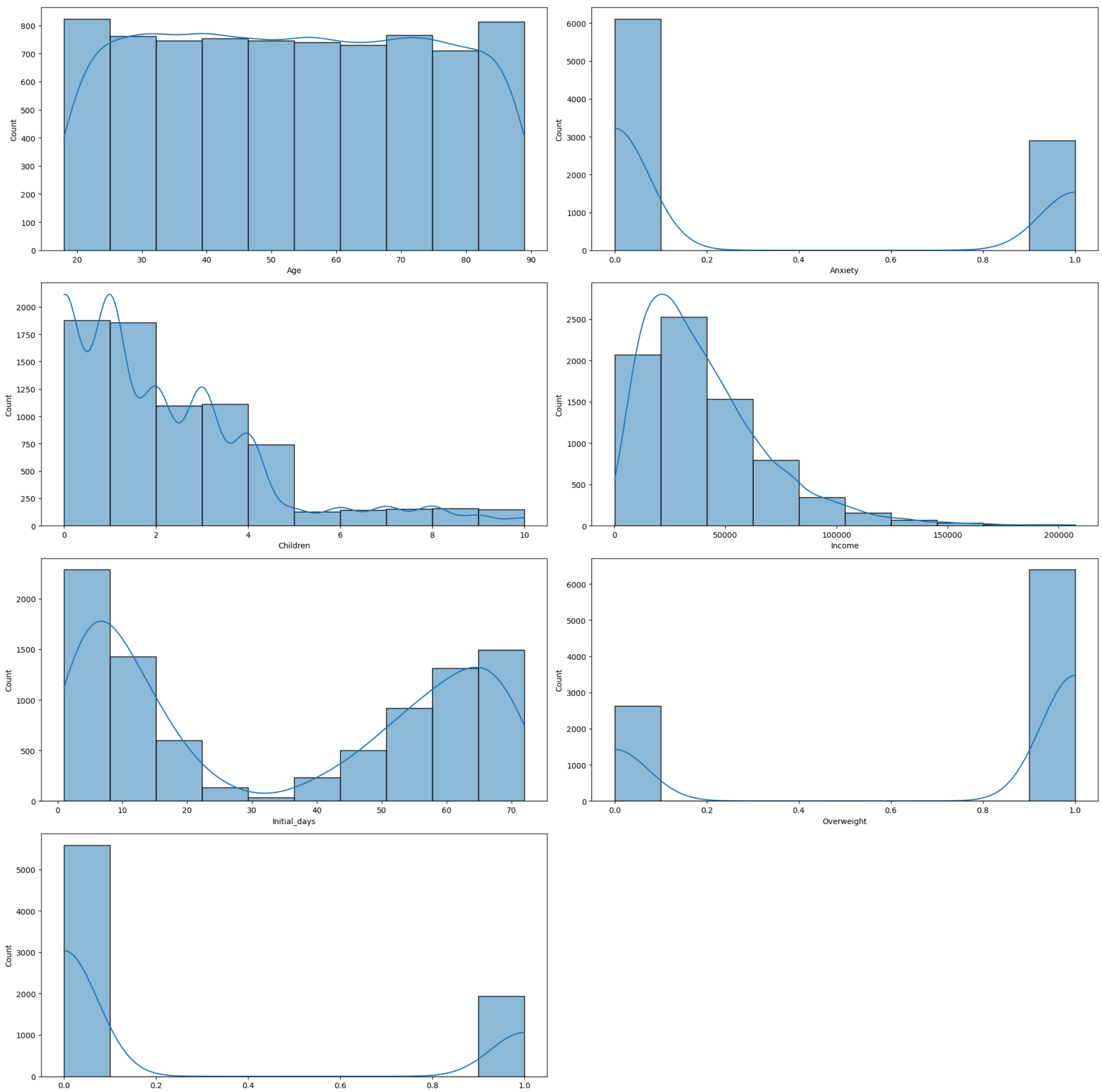
```
# Define values as numbers for imputation purposes. Method seen by Dr. Middleton,
dict = {"Soft_drink": {"No": 0, "Yes": 1, "unknown": np.nan}}
df_clean.replace(dict, inplace = True)

print('Missing Values:')

for col in df_clean.sort_index(axis = 1):
    if df_clean[col].isna().sum() > 0:
        print(col)
```

```
Missing Values:
Age
Anxiety
Children
Income
Initial_days
Overweight
Soft_drink
```

Examining Distribution:

In [116…

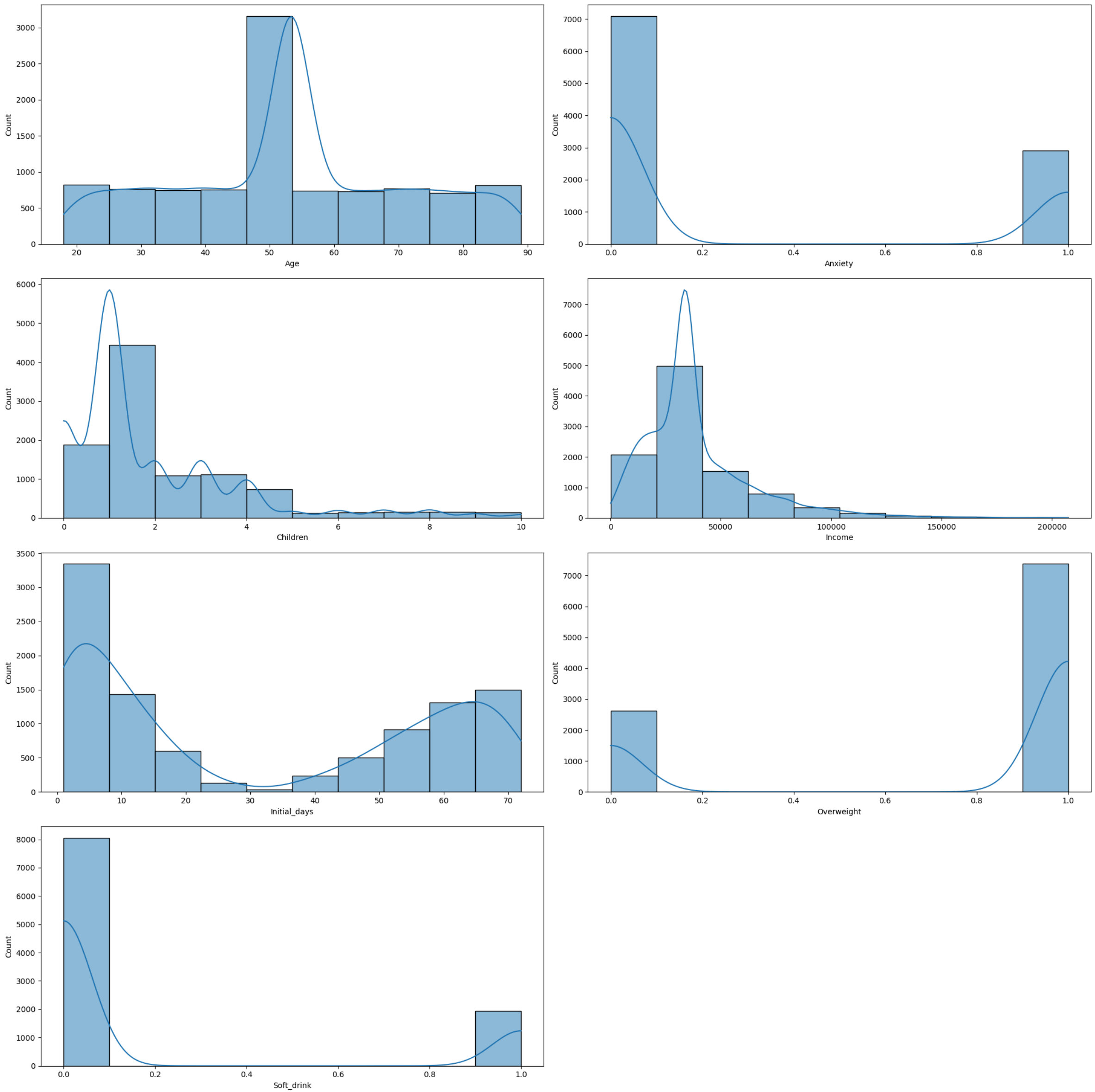
```
# Ignore FutureWarnings (from sns inf -> NaN.)
warnings.simplefilter(action='ignore', category=FutureWarning)

fig, ax = plt.subplots(4, 2, figsize = (20, 20))
ax = ax.flatten()

ax[-1].axis('off')

null_list = sorted(df.columns[df.isna().any()].tolist())

for i, col in enumerate(null_list):
    if (df_clean[col].dtypes != 'bool'):
        sns.histplot(df_clean[col], bins = 10, ax = ax[i], kde = True)

plt.tight_layout()
plt.show()
```



Perform Imputation:

In [117…

```
# Based on the distribution of the graphs, imputate.
# Uniform -> Mean, Skewed -> Median, Bimodial -> Mode.

mean_col = ['Age']
median_col = ['Children', 'Income']
mode_col = ['Anxiety', 'Initial_days', 'Overweight', 'Soft_drink']

age_before = df_clean['Age'].describe()
chi_before = df_clean['Children'].median()
inc_before = df_clean['Income'].median()

# Impute on columns with null (NaN values) data. Method seen by Dr. Middleton, Ge
for col in null_list:
    if col in mean_col:
        df_clean[col].fillna(df_clean[col].mean(), inplace = True)
    elif col in median_col:
        df_clean[col].fillna(df_clean[col].median(), inplace = True)
    elif col in mode_col:
        df_clean[col] = df_clean[col].fillna(df_clean[col].mode()[0])

age_after = df_clean['Age'].describe()
chi_after = df_clean['Children'].median()
inc_after = df_clean['Income'].median()

fig, ax = plt.subplots(4, 2, figsize = (20, 20))
ax = ax.flatten()

ax[-1].axis('off')

for i, col in enumerate(null_list):
    if (df_clean[col].dtypes != 'bool'):
        sns.histplot(df_clean[col], bins = 10, ax = ax[i], kde = True)

plt.tight_layout()
plt.show()
```



Verification:

In [118…

```
# Verify the imputation of data for null values. As seen in 'Video 2: Getting Sta
# Treating Missing Values'
print('Age:\t\t|\tBefore\t\t\tAfter')
print('Mean\t\t|\t' + str(age_before['mean']) + ',\t' + str(age_after['mean']) +

print('Children:\t|\tBefore\t\t\tAfter')
print('Median\t\t|\t' + str(chi_before) + ',\t\t\t' + str(chi_after) + '\n')

print('Income:\t\t|\tBefore\t\t\tAfter')
print('Median\t\t|\t' + str(inc_before) + ',\t\t' + str(inc_after))
```

```
Age:            |       Before                  After
Mean            |       53.29567624571579,      53.29567624571578

Children:       |       Before                  After
Median          |       1.0,                    1.0

Income:         |       Before                  After
Median          |       33942.28,               33942.28
```

Verification - Missing Values

In [119…

```
# Detect null values in dataset.
df_clean.isnull().sum().sort_values(ascending = False)
```

Out[119…

```
CaseOrder             0
Customer_id           0
HighBlood             0
Stroke                0
Complication_risk     0
Overweight            0
Arthritis             0
Diabetes              0
Hyperlipidemia        0
BackPain              0
Anxiety               0
Allergic_rhinitis     0
Reflux_esophagitis    0
Asthma                0
Services              0
Initial_days          0
TotalCharge           0
Additional_charges    0
Item1                 0
Item2                 0
Item3                 0
Item4                 0
Item5                 0
Item6                 0
Item7                 0
Initial_admin         0
Soft_drink            0
VitD_supp             0
Timezone              0
Interaction           0
UID                   0
City                  0
State                 0
County                0
Zip                   0
Lat                   0
Lng                   0
Population            0
Area                  0
Job                   0
Full_meals_eaten      0
Children              0
Age                   0
Education             0
Employment            0
Income                0
Marital               0
Gender                0
ReAdmis               0
VitD_levels           0
Doc_visits            0
Item8                 0
dtype: int64
```

Mitigating - Outliers

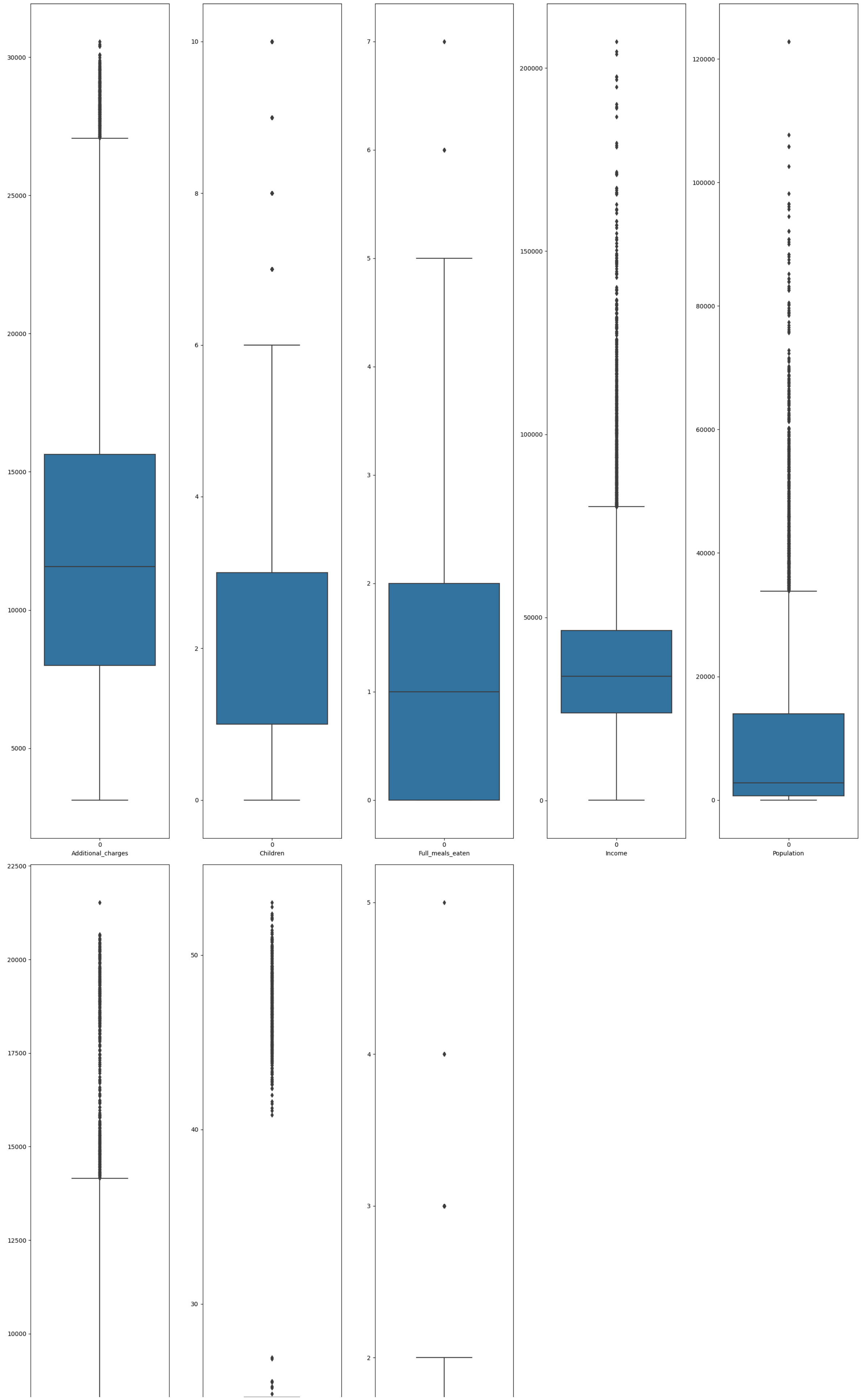
In [121…

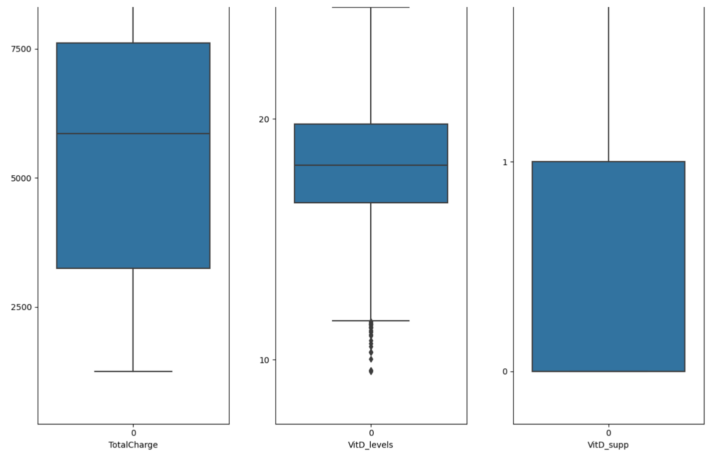
```
# Find which columns have outliers.
out_cols = []


for col in df_clean:
   if ((df_clean[col].dtypes == 'float64' or df_clean[col].dtypes == 'int64')) ar
        Q1 = df_clean[col].describe()['25%']
        Q3 = df_clean[col].describe()['75%']
        IQR = Q3 - Q1
        whiskerL = (Q1 - (1.5 * IQR))
        whiskerR = (Q3 + (1.5 * IQR))
        whiskerL = (df_clean[df_clean[col] >= whiskerL][col].min())
        whiskerR = (df_clean[df_clean[col] <= whiskerR][col].max())
        if any(df_clean[col] > whiskerR) or any(df_clean[col] < whiskerL):
            out_count = ((df_clean[col] > whiskerR).sum() + (df_clean[col] < whi
            out_cols.append(str(col))
        else:
            status = 'False'

print('Outliers:')
print(out_cols)
```

```
Outliers:
['Population', 'Children', 'Income', 'VitD_levels', 'Full_meals_eaten', 'VitD_sup
p', 'TotalCharge', 'Additional_charges']
```

In [122…

```
# Plot columns with outliers.
fig, ax = plt.subplots(2, 5, figsize = (20, 40))
ax = ax.flatten()

ax[-1].axis('off')
ax[-2].axis('off')

out_list = sorted(out_cols)

for i, col in enumerate(out_list):
    boxV = sns.boxplot(df_clean[col], ax = ax[i])
    boxV.set_xlabel(str(col))

plt.tight_layout()
plt.show()
```

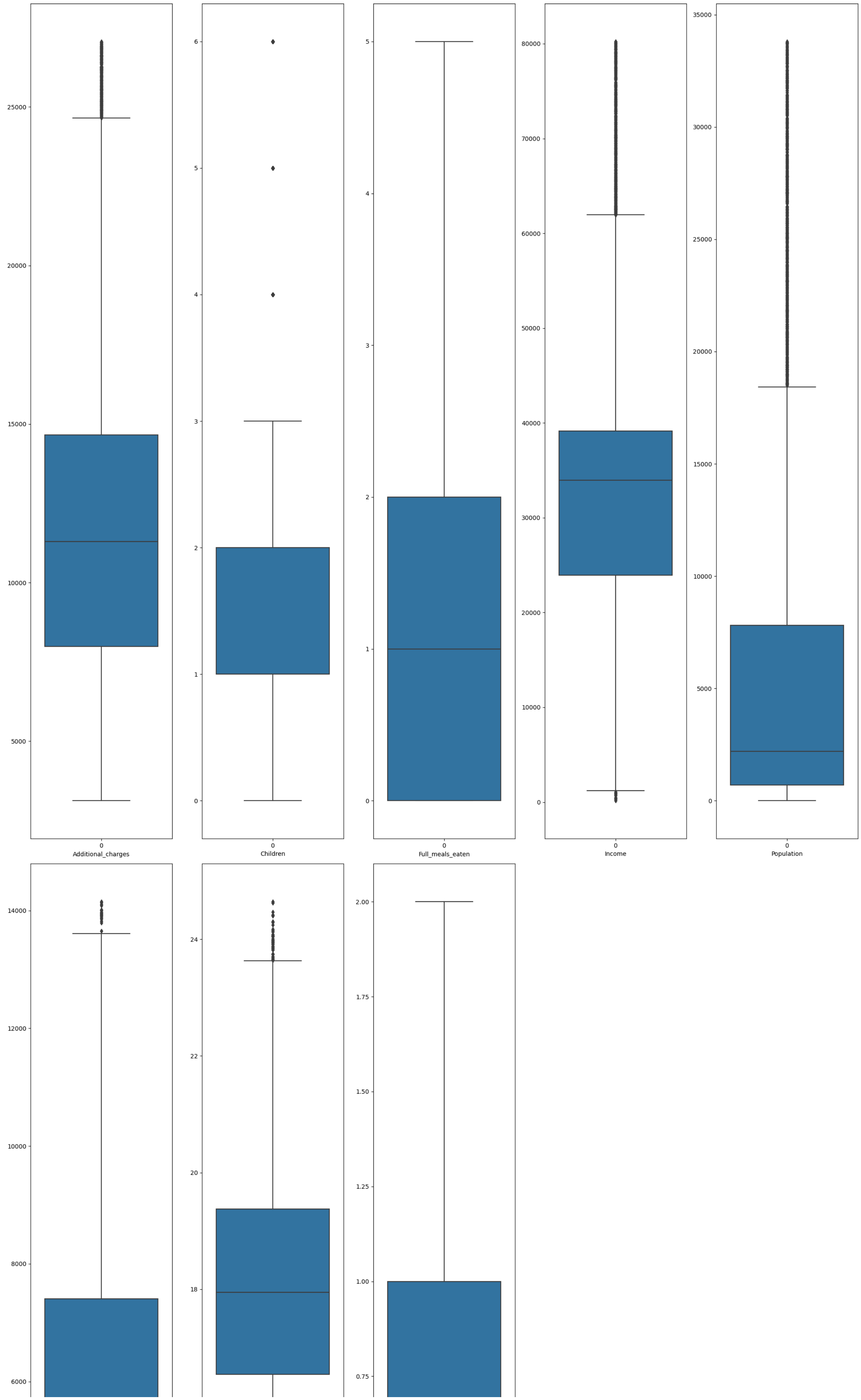

Additional_charges
Children
Full_meals_eaten
Income
Population

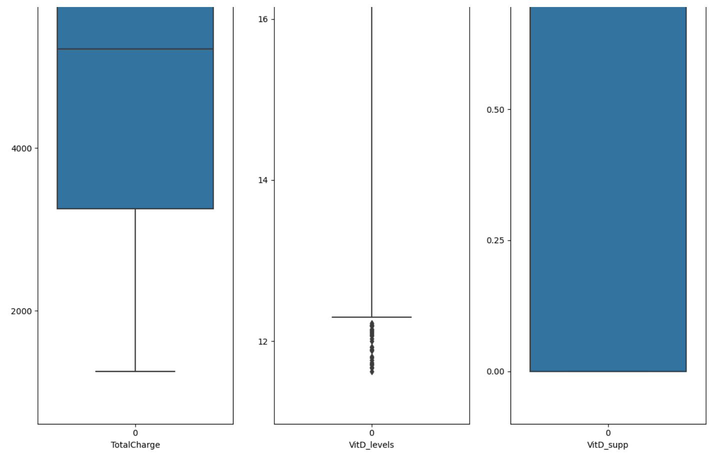


Verification - Outliers

In [123…

```
# If a column with data type 'float' or ' int', find its' whiskers and thus its'
# 'Video 3: Getting Started with D206 Detecting and Treating Outliers'
for col in df_clean:
   if ((df_clean[col].dtypes == 'float64' or df_clean[col].dtypes == 'int64')) an
        Q1 = df_clean[col].describe()['25%']
        Q3 = df_clean[col].describe()['75%']
        IQR = Q3 - Q1
        whiskerL = (Q1 - (1.5 * IQR))
        whiskerR = (Q3 + (1.5 * IQR))
        whiskerL = (df_clean[df_clean[col] >= whiskerL][col].min())
        whiskerR = (df_clean[df_clean[col] <= whiskerR][col].max())
        if any(df_clean[col] > whiskerR) or any(df_clean[col] < whiskerL):
            if (any(df_clean[col] > whiskerR) and not any(df_clean[col] < whisker
                df_clean[col] = np.where(df_clean[col] > whiskerR, np.nan, df_cle
                df_clean[col].fillna(df_clean[col].median(), inplace = True)
            elif (any(df_clean[col] < whiskerL) and not any(df_clean[col] > whisk
                df_clean[col] = np.where(df_clean[col] < whiskerL, np.nan, df_cle
                df_clean[col].fillna(df_clean[col].median(), inplace = True)
            else:
                df_clean[col] = np.where(df_clean[col] < whiskerL, np.nan, df_cle
                df_clean[col] = np.where(df_clean[col] > whiskerR, np.nan, df_cle
                df_clean[col].fillna(df_clean[col].median(), inplace = True)

fig, ax = plt.subplots(2, 5, figsize = (20, 40))
ax = ax.flatten()

ax[-1].axis('off')
ax[-2].axis('off')

out_list = sorted(out_cols)

# Set each boxplot with name of column it represents.
for i, col in enumerate(out_list):
    boxV = sns.boxplot(df_clean[col], ax = ax[i])
    boxV.set_xlabel(str(col))

plt.tight_layout()
plt.show()
```


Additional_charges
Children
Full_meals_eaten
Income
Population



Mitigating - Re-expression of Variables

In [124…

```
# Data frame before mitigation.
df_clean.T
```

Out[124…

	0	
CaseOrder	1	
Customer_id	C412403	Z91918
Interaction	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	d2450b70-0337-4406-bdbb bc1037f1734
UID	3a83ddb66e2ae73798bdf1d705dc0932	176354c5eef714957d486009feabf19
City	Eva	Mariann
State	AL	F
County	Morgan	Jackso
Zip	35621	3244
Lat	34.3496	30.8451
Lng	-86.72508	-85.2290
Population	2951.0	11303.
Area	Suburban	Urba
Timezone	America/Chicago	America/Chicag
Job	Psychologist, sport and exercise	Community development worke
Children	1.0	3.
Age	53.0	51.
Education	Some College, Less than 1 Year	Some College, 1 or More Years, N Degre
Employment	Full Time	Full Tim
Income	33942.28	46805.9
Marital	Divorced	Marrie
Gender	Male	Femal
ReAdmis	No	N
VitD_levels	17.80233	18.9946
Doc_visits	6	
Full_meals_eaten	0.0	2.
VitD_supp	0.0	1.
Soft_drink	0.0	0.
Initial_admin	Emergency Admission	Emergency Admissio
HighBlood	Yes	Ye

	0	
Stroke	No	N
Complication_risk	Medium	Hig
Overweight	0.0	1.
Arthritis	Yes	N
Diabetes	Yes	N
Hyperlipidemia	No	N
BackPain	Yes	N
Anxiety	1.0	0.
Allergic_rhinitis	Yes	N
Reflux_esophagitis	No	Ye
Asthma	Yes	N
Services	Blood Work	Intravenou
Initial_days	10.58577	15.12956
TotalCharge	3191.048774	4214.90534
Additional_charges	17939.40342	17612.9981
Item1	3	
Item2	3	
Item3	2	
Item4	2	
Item5	4	
Item6	3	
Item7	3	
Item8	4	

52 rows × 10000 columns

In [125…

```
# Regions used for 'State' one-hot encoding.
regionW = ["AK", "WA", "ID", "MT", "OR", "WY", "CA", "NV", "UT", "CO", "AZ", "NM
regionC = ["ND", "MN", "WI", "MI", "SD", "NE", "IA", "IL", "IN", "KS", "MO", "OK
regionE = ["ME", "NY", "VT", "NH", "MA", "CT", "RI", "OH", "PA", "NJ", "WV", "MD

# New columns made for one-hot encoding.
one_hot_cols = ['Blood_Work', 'CT_Scan', 'Central', 'Divorced', 'East', 'Electiv
                'MRI', 'Male', 'Married', 'Never_Married', 'No_Employment', 'Non
                'Separated', 'Student_Employment', 'Suburban', 'Urban', 'West',
```

```
for col in one_hot_cols:
    df_clean[col] = 0

# One-hot encoding, State:
for idx, row in df_clean.iterrows():
    currRow = row['State']

    if currRow in regionW:
        df_clean.loc[idx, 'West'] = 1
    elif currRow in regionC:
        df_clean.loc[idx, 'Central'] = 1
    elif currRow in regionE:
        df_clean.loc[idx, 'East'] = 1

# One-hot encoding, Area:
for idx, row in df_clean.iterrows():
    currRow = row['Area']

    if currRow == ('Suburban'):
        df_clean.loc[idx, 'Suburban'] = 1
    elif currRow == ('Urban'):
        df_clean.loc[idx, 'Urban'] = 1
    elif currRow == ('Rural'):
        df_clean.loc[idx, 'Rural'] = 1

# One-hot encoding, Employment:
for idx, row in df_clean.iterrows():
    currRow = row['Employment']

    if currRow == ('Full Time'):
        df_clean.loc[idx, 'Full_Time_Employment'] = 1
    elif currRow == ('Part Time'):
        df_clean.loc[idx, 'Part_Time_Employment'] = 1
    elif currRow == ('Retired'):
        df_clean.loc[idx, 'Retired'] = 1
    elif currRow == ('Student'):
        df_clean.loc[idx, 'Student_Employment'] = 1
    elif currRow == ('Unemployed'):
        df_clean.loc[idx, 'No_Employment'] = 1

# One-hot encoding, Marital:
for idx, row in df_clean.iterrows():
    currRow = row['Marital']

    if currRow == ('Divorced'):
        df_clean.loc[idx, 'Divorced'] = 1
    elif currRow == ('Married'):
        df_clean.loc[idx, 'Married'] = 1
    elif currRow == ('Never Married'):
        df_clean.loc[idx, 'Never_Married'] = 1
    elif currRow == ('Separated'):
        df_clean.loc[idx, 'Separated'] = 1
    elif currRow == ('Widowed'):
        df_clean.loc[idx, 'Widowed'] = 1
```

```
# One-hot encoding, Gender:
for idx, row in df_clean.iterrows():
    currRow = row['Gender']

    if currRow == ('Male'):
        df_clean.loc[idx, 'Male'] = 1
    elif currRow == ('Female'):
        df_clean.loc[idx, 'Female'] = 1
    elif currRow == ('Prefer not to answer'):
        df_clean.loc[idx, 'Nonbinary'] = 1


# One-hot encoding, Initial_admin:
for idx, row in df_clean.iterrows():
    currRow = row['Initial_admin']

    if currRow == ('Emergency Admission'):
        df_clean.loc[idx, 'EmergencyAdmission'] = 1
    elif currRow == ('Elective Admission'):
        df_clean.loc[idx, 'ElectiveAdmission'] = 1
    elif currRow == ('Observation Admission'):
        df_clean.loc[idx, 'ObservationAdmission'] = 1

# One-hot encoding, Services:
for idx, row in df_clean.iterrows():
    currRow = row['Services']

    if currRow == ('Blood Work'):
        df_clean.loc[idx, 'Blood_Work'] = 1
    elif currRow == ('Intravenous'):
        df_clean.loc[idx, 'Intravenous'] = 1
    elif currRow == ('CT Scan'):
        df_clean.loc[idx, 'CT_Scan'] = 1
    elif currRow == ('MRI'):
        df_clean.loc[idx, 'MRI'] = 1
```

In [126…

```
# Ordinal encoding, Education:
dict = {"Education": {"Some College, Less than 1 Year": 5,
            "Some College, 1 or More Years, No Degree": 5,
            "GED or Alternative Credential": 4, "Regular High School Diploma": 4
            "Bachelor's Degree": 7, "Master's Degree": 8,
            "Nursery School to 8th Grade": 2,
            "9th Grade to 12th Grade, No Diploma": 3, "Doctorate Degree": 9,
            "Associate's Degree": 6, "Professional School Degree": 9,
            "No Schooling Completed": 1}}

df_clean.replace(dict, inplace = True)

# Ordinal encoding, Complication_risk:
dict = {"Complication_risk": { "Low": 1, "Medium": 2, "High": 3}}
df_clean.replace(dict, inplace = True)

trsf_bool = ['Soft_drink', 'Initial_admin', 'HighBlood', 'Stroke', 'Complication
             'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic_rhin

# String boolean conversiom.
```

```
for col in trsf_bool:
    dict = {str(col): { "Yes": 1, "No": 0}}
    df_clean.replace(dict, inplace = True)

df_clean.replace(dict, inplace = True)
```

In [127…

```
# Column data type re-expression.
# List of columns to convert to bool
bool_cols = ['Soft_drink', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis', 'Dia
             'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Wes
             'Urban', 'Rural', 'Full_Time_Employment', 'Part_Time_Employment', 'I
             'No_Employment', 'Divorced', 'Married', 'Never_Married', 'Separated
             'Nonbinary', 'EmergencyAdmission', 'ElectiveAdmission', 'Observatio
             'CT_Scan', 'MRI', 'ReAdmis']

# Re-express as bool.
for col in bool_cols:
    df_clean[col] = df_clean[col].astype('bool')

int_cols = ['Population', 'Children', 'Age', 'Complication_risk', 'Item1', 'Item
            'Item5', 'Item6', 'Item7', 'Item8']

# Re-express as int64.
for col in int_cols:
    df_clean[col] = df_clean[col].astype('int64')

dec_cols = ['VitD_levels', 'Full_meals_eaten', 'VitD_supp', 'Initial_days',
            'TotalCharge', 'Additional_charges']

# Re-express precision.
for col in dec_cols:
    df_clean[col] = df_clean[col].round(2)
```

In [128…

```
# Remove the variables that were used for one-hot encoding.
df_clean = df_clean.drop(columns = ['City', 'State','County', 'Zip', 'Area', 'Em
df_clean.T
```

Out[128…

	0	
CaseOrder	1	
Customer_id	C412403	Z919
Interaction	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	d2450b70-0337-4406-bd bc1037f17
UID	3a83ddb66e2ae73798bdf1d705dc0932	176354c5eef714957d486009feabf
Lat	34.3496	30.84
...	...	
Student_Employment	False	F
Suburban	True	F
Urban	False	
West	False	F
Widowed	False	F

68 rows × 10000 columns

In [129…

```
# Re-assign data frame with order of old data frame with the inclusion of one-ho
df_clean = df_clean[['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'West', '
                'Timezone', 'Job', 'Children', 'Age', 'Education', 'Full_Time_Em
                'Income', 'Divorced', 'Married', 'Never_Married', 'Separated', '
                'Full_meals_eaten', 'VitD_supp', 'Soft_drink', 'EmergencyAdmissi
                'Overweight', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPa
                'CT_Scan', 'MRI', 'Initial_days', 'TotalCharge', 'Additional_cha
```

Verification - Re-expression of Variables

In [130…

```
# Verification of dataset post-mitigation.
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 68 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CaseOrder             10000 non-null  int64  
 1   Customer_id           10000 non-null  object 
 2   Interaction           10000 non-null  object 
 3   UID                   10000 non-null  object 
 4   West                  10000 non-null  bool   
 5   Central               10000 non-null  bool   
 6   East                  10000 non-null  bool   
 7   Lat                   10000 non-null  float64
 8   Lng                   10000 non-null  float64
 9   Population            10000 non-null  int64  
 10  Suburban              10000 non-null  bool   
 11  Urban                 10000 non-null  bool   
 12  Rural                 10000 non-null  bool   
 13  Timezone              10000 non-null  object 
 14  Job                   10000 non-null  object 
 15  Children              10000 non-null  int64  
 16  Age                   10000 non-null  int64  
 17  Education             10000 non-null  int64  
 18  Full_Time_Employment  10000 non-null  bool   
 19  Part_Time_Employment  10000 non-null  bool   
 20  Retired               10000 non-null  bool   
 21  Student_Employment    10000 non-null  bool   
 22  No_Employment         10000 non-null  bool   
 23  Income                10000 non-null  float64
 24  Divorced              10000 non-null  bool   
 25  Married               10000 non-null  bool   
 26  Never_Married         10000 non-null  bool   
 27  Separated             10000 non-null  bool   
 28  Widowed               10000 non-null  bool   
 29  Male                  10000 non-null  bool   
 30  Female                10000 non-null  bool   
 31  Nonbinary             10000 non-null  bool   
 32  ReAdmis               10000 non-null  bool   
 33  VitD_levels           10000 non-null  float64
 34  Doc_visits            10000 non-null  int64  
 35  Full_meals_eaten      10000 non-null  float64
 36  VitD_supp             10000 non-null  float64
 37  Soft_drink            10000 non-null  bool   
 38  EmergencyAdmission    10000 non-null  bool   
 39  ElectiveAdmission     10000 non-null  bool   
 40  ObservationAdmission  10000 non-null  bool   
 41  HighBlood             10000 non-null  bool   
 42  Stroke                10000 non-null  bool   
 43  Complication_risk     10000 non-null  int64  
 44  Overweight            10000 non-null  bool   
 45  Arthritis             10000 non-null  bool   
 46  Diabetes              10000 non-null  bool   
 47  Hyperlipidemia        10000 non-null  bool   
 48  BackPain              10000 non-null  bool   
 49  Anxiety               10000 non-null  bool   
 50  Allergic_rhinitis     10000 non-null  bool   
```

```
 51  Reflux_esophagitis    10000 non-null  bool
 52  Asthma                10000 non-null  bool
 53  Blood_Work            10000 non-null  bool
 54  Intravenous           10000 non-null  bool
 55  CT_Scan               10000 non-null  bool
 56  MRI                   10000 non-null  bool
 57  Initial_days          10000 non-null  float64
 58  TotalCharge           10000 non-null  float64
 59  Additional_charges    10000 non-null  float64
 60  Item1                 10000 non-null  int64
 61  Item2                 10000 non-null  int64
 62  Item3                 10000 non-null  int64
 63  Item4                 10000 non-null  int64
 64  Item5                 10000 non-null  int64
 65  Item6                 10000 non-null  int64
 66  Item7                 10000 non-null  int64
 67  Item8                 10000 non-null  int64
dtypes: bool(39), float64(9), int64(15), object(5)
memory usage: 2.6+ MB
```

D5: Clean Data

In [131…

```python
# Copy of clean dataset.
df_clean.to_csv('medical_raw_data_clean.csv')
```

D6: Limitations

The data-cleaning process involved a general lack of familiarity with the programming language, Python, used for cleaning introducing growing pains in the process. Additionally, inconsistencies among qualitative variables added ambiguity, which is hurtful for accuracy. The presence of missing values in three of the qualitative variables posed a unique challenge, as imputation for such a small range, 0 and 1, may skew the dataset innappropriately. Inappropriately formatted fields added extra steps to convert data into a usable format. A general lack of understanding in domain knowledge regarding hospital operations, may have hindered the ability to identify errors or anomalies within the dataset, possibly leading to incorrect decisions. These challenges define the importance of expertise in both the technical and subject matter aspects of the data-cleaning to ensure proper outcomes.

D7: Impact of Limitations

As mentioned previously, lack of Python familiarity involved growing pains which thus significantly slowed the cleaning process. Additionally, the lack of familiarity may have inhibited skipping out on other methods more effective for analysis. The missing values may be misrepresenting the entire picture of an observation. Imputation on boolean fields, also may misrepresent the entire picture of an observation. The lack of domain knowledge may limit the interpretation of the data and thus may not generate an accurate dataset for future use.

E1: Principal Components

In [132…

```
# Quantitative (continous) variables only.
pca_cols = df_clean[['Lat', 'Lng', 'Population', 'Children', 'Age', 'Income', 'V
                     'VitD_supp', 'Initial_days', 'TotalCharge', 'Additional_char

# Normalization to ensure PCA algorithm captures appropriate variance of data.
cols_normalized = (pca_cols - pca_cols.mean())/pca_cols.std()

# Sets the number of components, 13 in this case.
pca = PCA(n_components=pca_cols.shape[1])

# Used for dimension reduction, calculating the eigenvalues and eigenvectors.
pca.fit(cols_normalized)

# Stores the analysis into a local data frame.
med_pca = pd.DataFrame(pca.transform(cols_normalized), columns = ['PCA1', 'PCA2'
                                                                  'PCA9', 'PCA10
# Organizes columns and respective pca values.
loadings = pd.DataFrame(pca.components_.T,
                        columns = ['PCA1', 'PCA2', 'PCA3', 'PCA4', 'PCA5', 'PCA6
                        index=pca_cols.columns)

loadings
```

Out[132…

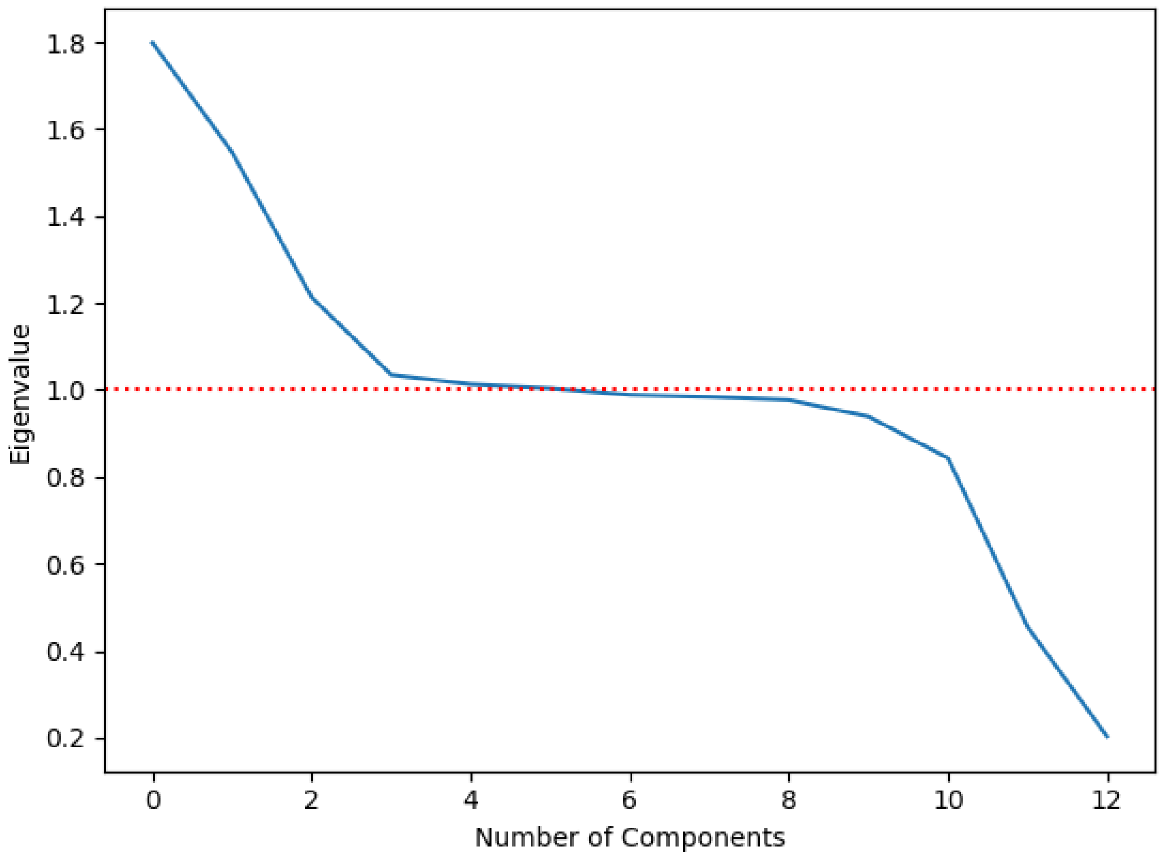
	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6	
Lat	-0.021960	-0.024889	0.639403	0.077157	-0.047315	0.104678	0.0
Lng	-0.005158	0.034415	-0.499811	-0.002214	0.015231	-0.088838	-0.1
Population	0.034536	-0.024141	-0.568801	-0.038217	0.033725	0.067249	0.0
Children	0.015640	-0.015352	-0.076238	0.015442	-0.153267	0.874886	0.2
Age	0.054333	0.703054	0.009155	0.031959	-0.027119	0.001147	0.0
Income	-0.002531	-0.015715	-0.037335	0.393293	0.500562	0.103205	0.5
VitD_levels	0.051246	0.025323	0.057529	-0.415653	0.577754	-0.041368	0.2
Doc_visits	-0.012482	0.010666	0.022327	0.234527	0.559705	0.289603	-0.7
Full_meals_eaten	-0.025165	0.036380	0.074515	-0.575919	0.241366	-0.051218	0.0
VitD_supp	0.045693	-0.005724	0.009823	0.525068	0.115414	-0.333846	0.1
Initial_days	0.700264	-0.065617	0.019272	0.009985	-0.048502	0.009059	-0.0
TotalCharge	0.704173	-0.045932	0.023025	-0.029218	0.022364	-0.003195	-0.0
Additional_charges	0.057989	0.703079	0.013032	0.023209	-0.010218	0.028291	0.0

E2: Criteria Used

Scree Plot

In [133…

```
# Principal Component Analysis. Code assistance via 'Getting Started with D206 |
cov_matrix = np.dot(cols_normalized.T, cols_normalized) / pca_cols.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for eigenv

# Plot the eigenvalues as assigned above.
plt.plot(eigenvalues)
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalue')
plt.axhline(y = 1, color = 'Red', linestyle = ':')

plt.tight_layout()
plt.show()
```



The Kaiser rule states that eigenvalues only greater than or equal to 1 are worth keeping. The graph depicts which components are over this limit though it is still hard to tell. The following code programmatically indicates that 6 variables follow the Kaiser rule. As seen in PCA1, Initial_days and TotalCharge may be the heaviest influence on PC1, Additional_charges on PC2, Lat on PC3 however and so on for the remaining eigenvalues greater than or equal to one as found below where generally, "...a component with an eigenvalue of 1 accounts for as much variance as single variable." (O'Connor, The number of eigenvalues greater than 1)

Kaiser Criterion:

In [134…

```
# Kaiser Rule.
for i, e in enumerate(eigenvalues):
    if e >= 1:
        print('PCA' + str(i + 1) + ': ' + str(e))
```

```
PCA1: 1.7979643111109211
PCA2: 1.546415888406625
PCA3: 1.213017447936713
PCA4: 1.0347603667253222
PCA5: 1.012824731704533
PCA6: 1.0036673420344593
```

E3: Benefits

Principal Component Analysis may benefit the company by means of dimension reduction and data visualization for data exploration. The data set does not have nearly as many continous quantitative variables as it does qualitative variables, however a correlation can be found with what results come from the PCA analysis from the thirteen provided. By transforming these relationships, as seen in the section before, 6 components may offer insight into the data provided. More specifically the relationship as seen between the TotalCharge and Initial_days, is perhaps more beneficial to the hospital than the initial scenario presents as PCA1 displays more variance than the variables by themselves. This dataset is particularly large and PCA thus can reduce such. The scope of what is important or beneficial to the hospital is indicitive of the business need and more.

Supporting Documents

F: Video

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=47162933-ce5f-41c2-bd72-b13b00436eb8

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d89e62c9-4527-4a98-ae29-b136014277cb (Only if interested, 40 min step by step demonstration.)

G: Acknowledgement of Web Sources

https://builtin.com/data-science/step-step-explanation-principal-component-analysis by Zakaria Jaadi
https://www.geeksforgeeks.org/principal-component-analysis-pca/
https://saturncloud.io/blog/how-to-get-column-name-which-contains-a-specific-value-at-any-rows-in-python-pandas/ by Saturn Cloud
https://learnpython.com/blog/sort-alphabetically-in-python/ by Xavier Riguolet

https://www.statology.org/pandas-isin-multiple-columns/ By "Zach"
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html by SciKit-Learn
https://seaborn.pydata.org/tutorial/function_overview.html by Seaborn

H: Acknowledgement of Sources

Dr. Eric Straw

Dr. Keiona Middleton, D206 Getting Started with D206 Videos/Slideshows

O’Connor, B. P. (n.d.). The number of eigenvalues greater than 1. R. https://search.r-project.org/CRAN/refmans/EFA.dimensions/html/NEVALSGT1.html

WGU Data Science Team

In [ ]:

```
```